CLAIMS

What is claimed is:

1. A method for dynamically managing a reassembly buffer, comprising:

providing a plurality of data blocks and an indirect list;

pointing, via entries in the indirect list, to allocated data blocks in the plurality of data blocks that currently store incoming data;

if a free data block in the plurality of data blocks is required for the storage of incoming data, allocating the free data block for storing incoming data; and,

if an allocated data block in the plurality of data blocks is no longer needed for storing incoming data, deallocating the allocated data block such that the deallocated data block becomes a free data block.

2. The method of claim 1, wherein the incoming data comprises TCP data.

3. The method of claim 1, further comprising a plurality of the indirect lists chained together.

4. The method of claim 3, wherein an entry in one of the plurality of indirect lists contains a pointer to another of the plurality of indirect lists.

5. The method of claim 1, wherein the plurality of data blocks comprises a pool of free data blocks, the method further comprising:

allocating and deallocating free data blocks from the pool of free data blocks using a free list, wherein the free list includes entries each containing a pointer to a free data block.

6. The method of claim 5, wherein the free list comprises a plurality of indirect lists chained together to form a stack.

7. The method of claim 5, further comprising:

providing a cache of free data blocks that are allocated before the free data blocks in the free list.

8. The method of claim 6, wherein the free list comprises a head pointer for pointing to an entry, in one of the plurality of indirect lists forming the free list, containing a pointer to a next available free data block to be allocated.

9. The method of claim 8, wherein:

if the head pointer points to an entry that is not a last entry in one of the plurality of indirect lists forming the free list, that entry contains a pointer to the next available free data block to be allocated; and

upon allocation of the next available free data block, the head pointer is moved to point to the next entry in the same indirect list.

10. The method of claim 8, wherein:

if the head pointer points to a last entry in one of the plurality of indirect lists forming the free list, that last entry contains a pointer to a first entry in a next indirect list in the chain of indirect lists forming the free list; and

the indirect list containing the last entry becomes a next available free data block to be allocated, and the head pointer is moved to point to the first entry in the next indirect list in the chain of indirect lists forming the free list.

11. The method of claim 6, wherein the free list comprises a tail pointer for pointing to an entry, in one of the plurality of indirect lists forming the free list, that will be assigned a pointer to a next allocated data block to be deallocated.

12. The method of claim 11, wherein:

if the tail pointer points to an entry that is not a last entry in one of the plurality of indirect lists forming the free list, updating the pointer of that entry to point to the next allocated data block to be deallocated; and

moving the tail pointer to point to a next entry in the same indirect list.

13. The method of claim 11, wherein:

if the tail pointer points to a last entry in one of the plurality of indirect lists forming the free list, that last entry is updated to contain a pointer to a first entry in the next allocated data block to be deallocated, which becomes a next indirect list in the in the chain of indirect lists forming the free list; and

the tail pointer is moved to point to the first entry in the next indirect list in the chain of indirect lists forming the free list.

14. The method of claim 1, further including:

selectively operating the reassembly buffer in a static mode by not deallocating allocated data blocks.

15. A system for dynamically managing a reassembly buffer, comprising:

a plurality of data blocks;

an indirect list having a plurality of entries; and

a memory manager for controlling allocation and deallocation of the plurality of data blocks;

wherein, if a free data block in the plurality of data blocks is required for the storage of incoming data, the memory manager allocates the free data block for storing incoming data; and wherein, if an allocated data block in the plurality of data blocks is no longer needed for storing incoming data, the memory manager deallocates the allocated data block such that the deallocated data block becomes a free data block.

16. The system of claim 15, wherein the incoming data comprises TCP data.

17. The system of claim 15, further comprising a plurality of the indirect lists chained together.

18. The system of claim 17, wherein an entry in one of the plurality of indirect lists contains a pointer to another of the plurality of indirect lists.

19. The system of claim 15, wherein the plurality of data blocks comprises a pool of free data blocks, wherein the memory manager allocates and deallocates free data blocks from

the pool of free data blocks using a free list, and wherein the free list includes entries each containing a pointer to a free data block.

20. The system of claim 19, wherein the free list comprises a plurality of indirect lists chained together to form a stack, and wherein the free list comprises:

a head pointer for pointing to an entry, in one of the plurality of indirect lists forming the free list, containing a pointer to a next available free data block to be allocated; and,

a tail pointer for pointing to an entry, in one of the plurality of indirect lists forming the free list, that will be assigned a pointer to a next allocated data block to be deallocated.

21. A computer program product stored on a recordable medium, which when executed, performs the method set forth in claim 1.

22. A method for storing out-of-order data segments in a reassembly buffer, comprising:

providing a plurality of data blocks and an indirect list having a plurality of entries;

providing each data segment with a sequence number, wherein the sequence number specifies which entry in the indirect list is to be associated with the data segment;

determining if any of the plurality of data blocks has already been allocated to the specified entry in the indirect list;

if a data block has already been allocated to the specified entry, storing the data segment in the allocated data block; and

if a data block has not already been allocated to the specified entry, allocating a free data block for the storage of the data segment, and storing the data segment in the allocated free data block.

23. The method of claim 22, wherein the step of determining if any of the plurality of data blocks has been allocated to the specified entry in the indirect list comprises:

providing each entry in the indirect list with a bit that indicates the allocated/free status of the entry; and

examining the bit to determine if any of the plurality of data blocks has been allocated to the specified entry in the indirect list.

24. The method of claim 22, wherein the sequence number specifies an offset in the

allocated data block for the storage of the data segment.

25. The method of claim 22, further comprising a plurality of indirect lists, wherein the sequence number specifies which of the plurality of indirect lists and which entry in the specified indirect list is to be associated with the data segment.